µC/OS-II – EIN ECHTZEITBETRIEBSSYSTEM FÜR MIKROCONTROLLER

Peter Balog

Fachhochschule Technikum Wien

ZUSAMMENFASSUNG:

µC/OS-II (MicroC/OS-II) ist ein optimierter, stark skalierbarer Multitasking- / Echtzeitbetriebssystemkern speziell für Mikrocontrolleranwendungen. Ein effektives prioritätsbasiertes, pre-emptives Task-Scheduling, zahlreiche Intertask Kommunikations- und Synchronisationsmethoden und eine einfache Speicherverwaltung, sowie die Fähigkeit ROM-fähige Applikationen zu realisieren, zeichnen µC/OS-II aus. Die umfangreiche Dokumentation, der verfügbare Quelltext und die zahlreichen Portierungen für unterschiedliche Hardware-Plattformen und Software-Entwicklungsumgebungen erklären die hohe Akzeptanz bei Embedded Systems Entwicklern.

EINLEITUNG

Das Echtzeitbetriebssystem, bzw. der Echtzeitbetriebssystemkern, µC/OS-II wurde speziell für Mikrocontrolleranwendungen von Jean J. Labrosse entwickelt und erstmals 1992 in der Vorgängerversion µC/OS veröffentlicht. Dem jüngsten Buch /1/ liegt die Version 2.52 von µC/OS-II zu Grunde. An der Fachhochschule Technikum Wien wird seit dem Wintersemester 2001/2002 mit µC/OS-II V2.51 auf C167-Plattformen mit der Keil µVision2 Entwicklungsumgebung in diversen „Embedded Systems – RTOS“ Lehrveranstaltungen gearbeitet.

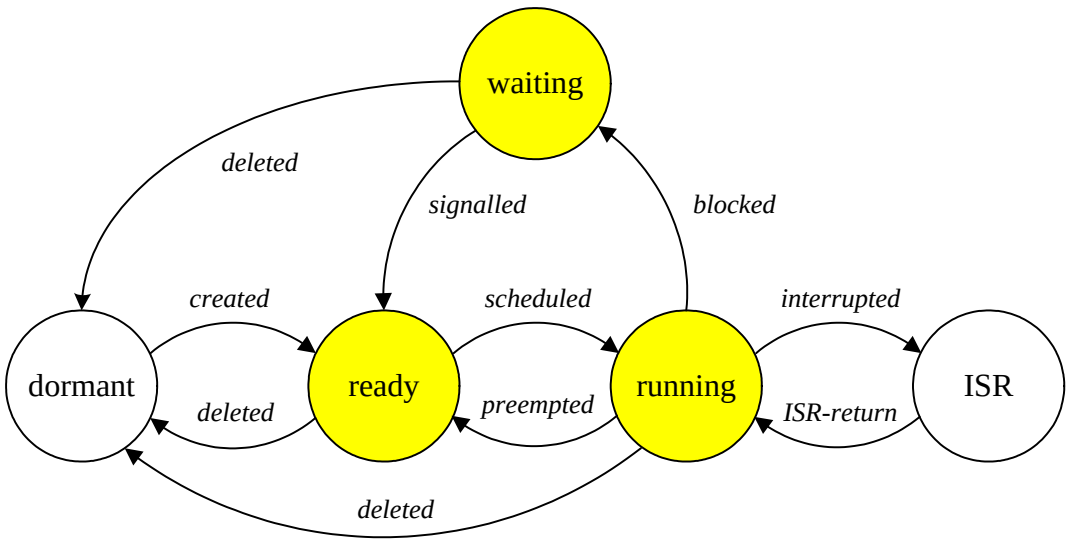
EIGENSCHAFTEN

µC/OS-II ist ein portabler, skalierbarer, pre-emptiver und ROM-fähiger Echtzeit- bzw. Multitaskingkern für Mikroprozessor- und Mikrocontrolleranwendungen. Eine µC/OS-II Applikation kann aus bis zu 56 parallelen Anwenderprozessen bestehen und auf folgende Betriebssystemdienste aufbauen:

- *Task Management*
- *Intertask Communication & Synchronization*
- *Time Management*
- *Fixed Sized Memory Block Management*

TASK KONZEPT

Der Betriebssystemkern kann 64 Tasks (parallele Prozesse) verwalten. Da 8 Tasks für interne Aufgaben reserviert sind, stehen dem Anwender 56 Applikations-Tasks zur Verfügung. In µC/OS-II wurde ein streng prioritätsbasiertes, pre-emptives Task-Scheduling implementiert. Jeder Task muss eine eindeutige Priorität (*priority number*) zugeordnet werden, anhand der auch die Task-Identifikation erfolgt. Die folgende Abbildung zeigt das zugrunde liegende Task-Zustandsdiagramm:



Gemäß der implementierten Scheduling-Strategie wird der höchstprioren Task die im Zustand *ready* ist, die CPU zugeteilt. Eine laufende Task (Zustand *running*) kann durch einen Interrupt unterbrochen werden; in der zugeordneten Interrupt-Service-Routine (ISR) erfolgt die entsprechende Verarbeitung. Im Zuge des Rücksprungs aus der ISR erfolgt wieder ein Task-Scheduling. Eine laufende Task kann weiters blockieren, wenn z.B. auf eine nicht vorhandene Ressource zugegriffen wird. Die Task geht dann in den Zustand *waiting* über und ein Task-Scheduling wird gestartet. Eine wartende Task wird wieder ablaufbereit (Zustand *ready*), wenn das Ereignis, auf das gewartet wird, eingetreten ist.

Eine neue Task kann jeder Zeit von einer laufenden Task kreiert werden. Eine Task im Zustand *ready* oder *waiting* kann von einer anderen (laufenden) Task gelöscht werden. Eine laufende Task kann nach erledigter Aufgabe selbst terminieren.

INTERTASK KOMMUNIKATION

In einer parallelen Applikation laufen mehrere sequentielle Prozesse (Tasks) parallel zu einander ab. Unabhängig von der Relation der Prozesse (nebenläufig, kooperierend) müssen sie i.A. Informationen austauschen oder einfach zeitlich bzw. aktionsabhängig synchronisiert werden. Es besteht somit ein Bedarf an Kommunikations- und Synchronisations-mechanismen. Der Betriebssystemkern µC/OS-II bietet die folgenden *Intertask Communication & Synchronization (ICS)* Objekte:

- *Semaphores*
- *Mutual Exclusion Semaphores*
- *Message Mailboxes*
- *Message Queues*
- *Event Flags*

Bei den µC/OS-II Semaphoren handelt es sich um klassische allgemeine Semaphore (*general semaphores, counting semaphores*) wie sie 1968 von E. W. Dijkstra /3/ vorgestellt wurden. Die *Mutual Exclusion* Semaphore hingegen sind spezielle binäre Semaphore zur Klammerung von kritischen Abschnitten zwischen nebenläufigen Prozessen, welche neben

der Realisierung des notwendigen wechselseitigen Ausschlusses auch noch das bekannte *Priority Inversion Problem* lösen, da sie temporär die Priorität des aufrufenden Prozesses erhöhen.

Die *Message Mailboxes* und *Message Queues* erlauben die Kommunikation von Nachrichten zwischen Tasks. Während eine *Mailbox* maximal eine Nachricht aufnehmen kann, verwaltet eine *Queue* im Prinzip eine Warteschlange von Nachrichten.

Event Flags können als Feld (*array*) von binären Semaphoren verstanden werden. Bei der Abfrage mit `OSFlagPend` bzw. `OSFlagAccept` besteht die Möglichkeit der logischen UND- und ODER-Verknüpfung von Flags; damit ist sowohl eine konjunktive als auch eine disjunktive Synchronisation möglich.

µC/OS-II unterscheidet zwischen Tasks, die vom Betriebssystem verwaltet werden, und Interrupt-Service-Routinen (ISR), welche außerhalb der Kontrolle des Betriebssystems stehen. Aus ISRs besteht jedoch die Möglichkeit an ICS-Objekte Signale zu senden, um so Tasks über externe Ereignisse (Interrupts) zu synchronisieren. Konsequent weiter gedacht bedeutet dies, dass in der ISR nur mehr die elementarsten Hardwareaktionen realisiert werden. Die eigentliche „Funktion zum Interrupt“ wird in eine Task gelegt.

Für jede ICS-Möglichkeit stellt das Betriebssystem einen Satz von Basisfunktionen zur Verfügung. Zusätzlich verfügen manche ICS-Mechanismen noch über Spezialfunktionen. Die Basisfunktionen sind:

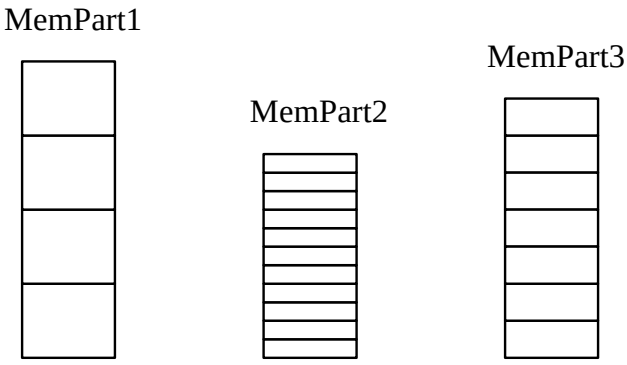
- *OSxxxCreate*
- *OSxxxDel*
- *OSxxxPend*
- *OSxxxAccept*
- *OSxxxPost*
- *OSxxxQuery*

Mit `OSxxxCreate` wird das entsprechende ICS-Objekt erzeugt, mit `OSxxxDel` kann es wiederum gelöscht werden. Mit `OSxxxPend` und `OSxxxAccept` wird ein ICS-Objekt abgefragt. Während `OSxxxAccept` auf jeden Fall zur aufrufenden Task zurückkehrt

blockiert `OSxxxPend` die aufrufende Task wenn die abgefragte Ressource nicht verfügbar ist. Die blockierte Task (Zustand *waiting*) wird wieder ablaufbereit (Zustand *ready*) wenn entweder die ICS-Ressource verfügbar ist oder ein optionales Timeout abgelaufen ist. Die Funktion `OSxxxPost` erlaubt das Signallisieren an ein ICS-Objekt. Um das Debugging zu erleichtern steht die Funktion `OSxxxQuery` zur Verfügung, mit der die µC/OS-II internen ISC-Verwaltungsdaten eingesehen werden können.

SPEICHERVERWALTUNG

µC/OS-II stellt eine einfache und zugleich effektive Speicherverwaltung zur Verfügung. Eine Applikation definiert eine entsprechende Anzahl von so genannten Speicherpartitionen, für die die Startadresse, die Blockgröße und die Anzahl der Blöcke spezifiziert werden. Über Betriebssystemfunktionen `OSMemGet` und `OSMemPut` können dann Speicherblöcke aus den zuvor definierten Partitionen angefordert bzw. retourniert werden.



SKALIERBARKEIT

Eine wesentliche Eigenschaft von µC/OS-II ist der hohe Grad an Skalierbarkeit, welcher besonders für Mikrocontroller basierte Applikationen von Vorteil ist. Das Konfigurations-management erfolgt über die Datei OS_CFG.H mit der nicht nur die Anzahl bzw. Größe von diversen Betriebssystemobjekten, sondern auch für verwendete Ressourcen die Verfügbarkeit auf Funktionsebene definiert werden kann. Werden z.B. für *Mutual Exclusion Semaphore* nur

die Funktionen `Pend` und `Post` benötigt, so kann der Code für die Funktionen `Accept`, `Del` und `Query` „ausgeblendet“ werden.

Damit kann die Speicherbelegung (*memory footprint*) je nach Anwendung flexibel optimiert werden.

PERFORMANCE

Die Ausführungszeit der meisten µC/OS-II Funktionen ist sowohl konstant als auch deterministisch, was bedeutet, dass die Ausführungszeit unabhängig von der Anzahl der parallelen Tasks der Applikation ist. Weiters ist µC/OS-II geeignet für sicherheitskritische Anwendungen im Luftfahrt- und Medizinbereich (FAA zertifiziert) und erfüllt zu 99% die C-Kodierungsstandards der MISRA (*Motor Industry Software Reliability Association*).

VERFÜGBARKEIT

Der Quelltext zur Version 2.52 von µC/OS-II ist in dem Buch /1/ veröffentlicht und sehr detailliert erklärt. Updates, Upgrades, etc. sind über die Internet-Homepage /2/ verfügbar. Für Ausbildungszwecke ist die uneingeschränkte Verwendbarkeit gegeben. Ausbildungsinstituten wird auch das jeweils aktuelle Quelltext-Paket (dzt. Version 2.70) zur Verfügung gestellt. Kommerzielle Anwender müssen µC/OS-II entsprechend lizenzieren (Details siehe /2/).

Über die Internet-Homepage /2/ stehen eine Vielzahl von µC/OS-II Portierungen für diverse Mikroprozessoren und Mikrocontroller, sowie Entwicklungsumgebungen, zum freien Download zur Verfügung. Für noch nicht portierte Hard- und Softwareplattformen werden im Kapitel „Porting µC/OS-II“ /1/ die notwendigen Schritte erklärt.

LITERATUR

1. Jean J. Labrosse, "MicroC/OS-II The Real-Time Kernel". CMP Books, 2nd Edition, ISBN 1-57820-103-9
2. http://www.ucos-ii.com
3. E. W. Dijkstra, „Co-operating sequential processes“, in „Programming Languages“ ed. F. Genuys, Academic Press London – New York 1968, pp 43-112